Bridging from Systems Engineering to Model Driven Development

Collaboration through Architecture

version 1.4
9/19/11
Welcome

• Key architectural techniques and work products serve as an effective bridge between systems engineering and software development
• This seminar outlines an approach for building this bridge and keeping it healthy
  – Focused on effective business and technical results
  – Joint engineering with a collaboration focus
  – Based on proven modeling disciplines, best practices
• Transform your separate Systems Engineering and Software Development groups into an efficient and effective joint team
Architecture as an Effective Bridge

- Challenge: Information Chasm
- Model Driven Engineering
- Collaboration: Approaches and Artifacts
- Business and Technical Results
• **Systems Engineering** is an engineering discipline producing
  – A high level description of the system
  – Requirements for functional behavior, and other characteristics
  – An overall system architecture, potentially in multiple aspects (logical software, electrical, mechanical, etc)
  – A decomposition of the system into components and sub-components, establishing interaction messages and protocols

• Effective systems engineering work products are
  – Clear, complete, consistent
  – Thorough, with a clear focus on the required capabilities
  – Detailed, but free of implementation bias
Software Development

- **Software Development** is an engineering discipline producing
  - An architecture for the system software
  - A decomposition of the software into components and sub-components, establishing interaction messages and protocols
  - *Actual, executable, deliverable software*
- **Effective software elements and components are**
  - Clear, consistent, maintainable
  - Complete, robust, satisfying all requirements
  - Flexible and long-lived, and durable in the face of changing requirements
Challenge: Information Chasm

• Effective collaboration between Systems Engineering and Software Development can
  – Reduce overall engineering costs and time to deliver product
  – Improve the robustness and overall quality of the system
• **But** – collaboration is generally awkward - each typically
  – Responds to different business and technical drivers
  – Uses distinct languages/tools to create work products
  – Focuses on different time horizons
  – Is subject to different measures of effectiveness

• Information transfer between the two groups is clumsy

• Creative input and overall control are a struggle
Challenge: Information Chasm

• Before effective collaboration can be established, each discipline must firmly base its own activities
  – Use clear and precise language with rigorous semantics
  – Follow sound, repeatable engineering principles – a methodology
  – Incorporate industry best practice
  – Build work products with their consumer in mind
  – Maintain focus on critical business performance factors
    • Productivity
    • Quality
    • Manageability/Predictability
Bridging from Systems Engineering to Model Driven Development: Collaboration through Architecture

Challenge: Information Chasm

- Graphical modeling languages provide
  - Abstract form of expression – a basis in models
  - Opportunity for uniformity of expression, rigor
  - Standards-based approaches
  - Rich, modern tools foundation with attendant automation

- Rigorous, model-based approaches provide a solid foundation for each discipline
  - Documented, teachable, manageable steps and techniques
  - Based on sound engineering and proven best practice
  - Produce known results on a repeatable basis
Challenge: Information Chasm

• A solid methodological base provides a firm footing
  – Central focus: Model Driven Approach
    • Integrated, self-consistent models based on known semantics
    • Diagrams that present specific views of the underlying model
    • Known to deliver specific business results
  – Following Iterative and Incremental Development
  – For Systems Engineering, maintain a strong focus on
    • Required work products, with a consumer-oriented perspective
    • Building a living, durable and flexible model of the system, its components and architecture
  – For Software Development, Models are best when they
    • Readily deploy to a range of implementation topologies - Platform Independent
    • Facilitate appropriate automation for document and code production
• With proven, model-based methodologies, we can start to establish collaboration at all levels
  – Individual model element
  – Work product
  – Engineering technique/process
  – Cultural
Challenge: Information Chasm

- The general collaborative identified here can be applied with a range of model driven methodologies that share key characteristics
  - Integrated models based on standard semantics
  - Platform Independence and emphasis on flexible, durable architecture
  - Use of automation of document production and code generation
- Your organization may already have these in place
• The OMG **Systems Modeling Language** (SysML) is a derivation of the UML for systems engineering
  – Specify
  – Analyze
  – Design
  – Verify
• **SysML provides a semantic foundation for modeling**
  – System Requirements
  – Behavior
  – Structure
  – Integration with a broad range of engineering analysis

• **For complex systems that may include**
  – Hardware
  – Software
  – Information
  – Personnel
  – Procedures
  – Facilities
• SysML is a subset of UML 2 with extensions
Models in Systems Engineering

- Key primitives include
  - Block
  - Flow
  - Requirement
  - Use Case
  - Parametric Constraint
  - Value Type
Some primitives are common to Software Development
Models in SW Development

- Model driven systems engineering can flow into model driven software development (MDD)
- MDD includes a wide range of techniques and technologies
- Model Driven Architecture is an advanced form of MDD
  - Architecture-focused
  - High degree of automation and optimization
  - Standards based and built upon open technology
Logical Architecture

• Partition a product or product family into logical components - subject matter *domains*:
  – Separate, conceptual universe
  – Inhabited by a related set of abstractions
  – Pertaining to a distinct subject matter
  – Interacts with other domains through published services
  – Independent of specific implementation topologies/distribution
Logical Architecture

- Logical Architecture is derived from the composition of the problem-space subject matters
Logical Architecture

• Includes:
  – A set of Logical Components - *Domains*
  – Arranged hierarchically – more abstract, executive level at the top. Less abstract, more implementation level at the bottom.
  – A set of sequence charts showing how the components work together to deliver the required capabilities of the system
  – Other supporting documentation
System Scenarios

- **System Scenario Models** show the allocation of behavioral responsibility among components:
• The **Class Model** shows a view of key elements within a logical component (domain):
  • Classes belonging to domain
  • Attributes - characteristics of a class
  • Operations – class behavior (state-independent)
  • Associations - relationships between classes
  • Generalization - hierarchies
  – Class Models are readily understandable by non-developers
  – Facilitates communication between all stakeholders
• UML Class Diagram:

- Entity Tracking
  - NewObservation()
  - DefineClient()
  - SelectTrackingSpecification()
  - EntityApproaching()
  - Identify()

- TrackerNode
  - «Identifier» trackerNodeId: Integer
  - reportNewTrackSvc: IncidentHandle
  - clientAddress: Integer
  - reportUpdateTrackSvc: IncidentHandle
  - reportOutOfViewSvc: IncidentHandle

- CorrelationRating
  - overlapPercentage: Real

- Observation
  - timestamp: Integer
  - quality: Real
  - location: gg_space_t
  - correlate()

- PredictedZone
  - locationZone: gg_space_t
  - staleTime: Integer

- Entity
  - «Identifier» reportId: Integer
  - reportable: Boolean
  - observationCount: Integer
  - correlateObservation()
  - updatePrediction()
  - addObservation()
  - selectTrackingSpecification()

- TrackingSpecification
  - «Identifier» priority: Integer
  - allowableStateTime: Integer
  - minimumObsQuality: Integer
  - verifyQuantity: Integer
  - timeoutValue: Integer
  - observationQueueSize: Integer
  - trackSpecId: Integer
State Machines

- **State Models:**
  - Define the lifecycle for a Class
  - Outline the Actions that a Class instance performs in response to Events
Collaboration

• Given model based systems engineering and software development, how can teams collaborate?
• Build a *platform* for collaboration
  – **Methodology**
    • Aligned along goals and general approach
    • Using common techniques where practical
  – **Language**
    • Derived from common and compatible base
    • Shared elements where possible
  – **Technology**
    • Common tools and repository
    • Open, extensible to facilitate appropriate, shared automation
Collaboration

• Focus this platform for collaboration through Architecture
  – **Methodology**: Techniques centered around the creation and evolution of the system architecture as the central conceptual foundation
  – **Language**: Primarily at the problem-space level of abstraction
  – **Technology**: Complete, rigorous and executable models at the analysis (problem space) level

• This platform brings your focus of development to your **Durable Architecture**
**Durable Architecture**: the key to sustained and effective collaboration

- A shared, sustained core focus of systems and software engineering
- Based on a problem-space focused level of abstraction
- Provides the canonical source for all model-specified information composing many work products
  - Systems Engineering Architecture and Design Documentation
  - Interface Specifications
  - Software Architecture and Design Documentation
  - Test Harnesses
  - Implementation Topology Configuration
• Durable Architecture is
  – A set of architecture-level models – some shared, some separate
  – An collaborative approach for their construction, evolution and joint use
  – A mindset for approaching model-based engineering, where problem-space-oriented models are the primary focus

• In contrast, static architecture
  – Is a by-product of other engineering efforts
  – Is a short-term/transitory output to satisfy external requirements
  – Stands apart from the “real system” and becomes irrelevant over time

• Durable Architecture is where collaboration happens
Collaboration - Durable Architecture

• These elements are **Jointly Developed**
  – Systems and Software participation throughout the lifecycle
  – Shared elements from single common repository
  – Carefully managed change control policy

• Same set of model artifacts is used equally on both sides

**MDSD Systems Engineering Activities**

<table>
<thead>
<tr>
<th>Rqts</th>
<th>Operational Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Cases</td>
<td>System Interfaces and External Interactions</td>
</tr>
<tr>
<td>Logical Architecture and Components</td>
<td>Physical Architecture, Component Interfaces and Functional Blocks</td>
</tr>
<tr>
<td>Detailed Behavior</td>
<td>System Data Models</td>
</tr>
</tbody>
</table>

**MDA Software Development Activities**

<table>
<thead>
<tr>
<th>Use Cases</th>
<th>Logical Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Scenarios</td>
<td>System Scenarios</td>
</tr>
<tr>
<td>Component Scenarios</td>
<td>Behavior</td>
</tr>
<tr>
<td>Actions</td>
<td>Dynamic Verification</td>
</tr>
<tr>
<td>Implementation Architecture</td>
<td>Integration</td>
</tr>
</tbody>
</table>
These elements are

- Jointly developed with Systems Engineering lead
- Critical input/review role by Software development
- Carefully managed change control policy

Partial set of model artifacts flow directly into software.
• The remaining elements **flowdown**
  – Created by Systems Engineering for Software Development (and others)

• Used as appropriate, with allocations to development elements, data carry forward, test input, general input, etc.
Collaboration Modeling

- Process keys to enable collaboration
  - Explicitly identify work products to be produced
    - Who is the consumer? (customer, development, SI&T, ?)
    - What are their needs for the work product?
    - What is the most efficient way to meet those needs?
  - Identify what model elements feed into what work products
    - Use automated generation to build/link efficiently
    - Express one fact in one place
    - Avoid manual maintenance of model-derived artifacts
  - For joint-developed and shared model elements
    - Ensure appropriate parties participate
    - Manage versions, reviews/updates, sharing explicitly
Technology keys to enable collaboration
- Common Integrated Development Environment (IDE)
- Common modeling toolset, repository
- Open, extensible environment to facilitate appropriate, shared automation
- Complete, rigorous and executable (when appropriate) models at the analysis (problem space) level
- Transformation to efficient, optimized implementations
Business Results

- Productivity
- Quality
- Longevity & Flexibility
- Risk Mitigation
Productivity

• Effective collaboration brings efficiency to the joint systems engineering/software development effort
  – Avoid duplicated efforts
  – Coordinate on areas impacting both disciplines, synchronize all aspects of the development lifecycle
    • Initial specification
    • Creation of detailed elements
    • Review, testing and refinement
    • Maintain and extend
  – Bringing the right experts in at each phase rapidly resolves issues as they arise – not downstream in later phases
Quality

• Bringing systems engineering and software development together establish a state of practice where
  – Seams between groups are minimized to eliminate responsibility and accountability gaps
  – Much higher degree of visibility is possible overall, ensuring thorough treatment of all components and process steps
  – The joint goals for a successful and quality product have a context to grow and thrive

• Specific collaborative engineering practices
  – Proactively reduce communication issues and inter-component alignment issues
  – Break down barriers between teams and foster cooperation
  – Increase fidelity between requirements and the delivered product
Longevity & Flexibility

- This type of inter-disciplinary collaboration produces systems with far greater longevity and flexibility
  - System architecture has a high degree of connection to software architecture
  - Improved traceability of requirements to software components and elements
  - Shared-concepts of Logical Architecture focus on durable aspects of the problem space, and not on transient deployment details
  - Reduced maintenance effort and turn-around time
  - Collaborative joint teams can assess and respond better to future needs
Risk Mitigation

- Key risks along the systems/software boundary are hard to address in isolation
  - Specifying an inconsistent system
  - Building the wrong system
  - Poor system performance
  - Brittle system architecture
  - System integration risks
- Collaborative process and joint teams mitigate key risks
  - Higher individual work product and overall system quality
  - Faster turnaround time for initial development and updates
  - Better end-to-end engineering fidelity: concept to code
Risk Mitigation

• **Skip Trial-and-Error self education with Systems Engineering Mentoring**
  
  – Apply specific expertise in critical areas
    
    • Model Driven Systems Engineering approach using UML or SysML
    
    • Specific optimizations to improve collaboration and system architecture longevity
  
  – Accelerate your adoption of model-based techniques
  
  – Look for a methodology with a rigorous and coherent set of steps
    
    • Collaboration focused
    
    • Based on proven modeling disciplines
Risk Mitigation

- Manage development risk with MDD/MDA Software Development Mentoring
  - Seek specialized experts in the rapid adoption of effective MDD/MDA methods and technology
- This type of expertise transfer can bring
  - Proven, best practice techniques for model driven development of complex, high performance systems
- Learn, Apply, Master
• **A proven mitigator for process change and technology risks:** engage an experienced guide
  
  – Pathfinder Solutions is a leading provider of model-driven technology, training and expertise for high performance, embedded, and real-time systems development:
  
  – We help you address the range of risks through training, mentoring, tools, and components for systems engineering, software architecture and development.
  
  – Sustained and repeatable, substantial business results
    
    • 2X productivity gains
    • 90% defect reduction
• Build a platform for collaboration through Architecture
  – **Methodology**: Techniques centered around the creation and evolution of the system architecture as the central conceptual foundation
  – **Language**: Primarily at the problem-space level of abstraction
  – **Technology**: Complete, rigorous and executable models at the analysis (problem space) level

• This platform brings your focus of development to your *Durable Architecture*

• This focus can bring dramatic results in
  – Productivity
  – Quality
  – Longevity & Flexibility
  – Risk Mitigation
Contact Us

Peter Fontana: 888-662-7284 x111
peterf@pathfindersolns.com

Bart Shappee: 888-662-7284 x116
barts@pathfindersolns.com